Discovering dynamics with genetic programming

Sašo Džeroski and Igor Petrovski

Institut Jožef Stefan, Jamova 39, 61111 Ljubljana, Slovenia

Abstract. This paper describes an application of the genetic programming paradigm to the problem of structure identification of dynamical systems. The approach is experimentally evaluated by reconstructing the models of several dynamical systems from simulated behaviors.

1 Introduction

The task of identification of dynamical systems (discovering dynamics), as addressed in this paper, can be defined informally as follows: Given an example behavior of a dynamical system, find a set of laws that describe the dynamics of the system. More precisely, a set of real-valued system variables is measured at regular intervals over a period of time, as illustrated in Table 1. The laws to be discovered (also called model of the dynamical system) are typically a set of differential equations $dX_i/dt = f_i(X_1, \ldots, X_n)$, $i = 1, \ldots, n$.

Table 1. A behavior trace of a dynamical system.

Time	System variables			
	X_1	X_2	$\ldots$	X_n
t_0	x_{10}	x_{20}	$\ldots$	x_{n0}
$t_1 = t_0 + h$	x_{11}	x_{21}	$\ldots$	x_{n1}
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$t_N = t_0 + Nh$	x_{1N}	x_{2N}	$\ldots$	x_{nN}

In mainstream system identification, as summarized by Ljung [4], the assumption is that the model structure (i.e. the functional form of each f_i) is known. The task is then to determine suitable values for the parameters in the model. This task is accordingly called *parameter identification*. In practice, many different model structures are tried out and the process of identification really becomes the process of evaluating and choosing between the resulting models in these different structures [4]. We refer to this task as *structure identification*.

Genetic algorithms [2] can be used to optimize the values of parameters in a fixed model structure, operating on parameter values encoded as bit strings. Genetic programming [3], on the other hand, can operate on populations of models (expressions) of different structure, which can also contain parameters. While genetic algorithms can be used for parameter identification, genetic programming can be used for structure identification.

In this paper, we describe the application of genetic programming to the problem of structure identification of dynamical systems. Section 2 presents in detail the implementation of the genetic programming paradigm intended for structure identification. The experimental evaluation of the approach is described in Section 3. Finally, Section 4 gives a brief discussion and concludes.

2 Structure identification with genetic programming

In the following, we describe our implementation of the genetic programming paradigm intended for discovering dynamics (GPDD). `Parameters` will be written in `teletype` font and their default values given in brackets.

An initial population of `pop_size` (300) individuals is first randomly generated. For a predefined number of `generations` (30) the population is evolved in the following manner. The fitness of each individual is calculated and track is kept of the individual with best fitness seen so far. The new population is created from the current one by applying fitness-proportionate reproduction and crossover. `perc_crossover` % (90 %) of the new population is formed by crossover and (100 - `perc_crossover`) % (10 %) by reproduction. Offspring that are already in the new generation or are too large (i.e. have more than `max_nodes` (30) nodes) are not inserted in the new population. Finally, after the predefined number of generations, the fittest individual (over all generations) is returned.

During the generation of the initial population, trees are randomly generated by choosing internal nodes from the set of available operators and terminal (leaf) nodes from the set of system variables. Terminal nodes can also be constants. `perc_full_tr` % (50 %) of the generated trees are full (i.e. have all leaves at the same depth), with depth ranging from 1 to `size_tr` (3). Operators are chosen according to a pre-specified probability distribution (default operators $\{+xy, -xy, /xy, *xy, \texttt{square}(x)\}$, probabilities $(0.24, 0.24, 0.24, 0.24, 0.04)$).

When generating random (non-full trees) an internal node is created with probability `perc_node_tr` % (40 %), and a terminal node (leaf) otherwise. `perc_const` % (50 %) of the leaf nodes are constants, `perc_blocks` % (20 %) of the remaining are building blocks if any are provided, and the rest are variables. We discuss building blocks below. Constants are assigned random initial values from the interval $[-5, 5]$; their values are optimized during the calculation of the fitness. Finally, duplicates are not inserted in the initial population.

Building blocks are subexpressions that can appear in the model sought. The user can specify them as a kind of background knowledge. They may contain generic variables (standing for any system variable), system variables, generic constants and operators. The values of the generic constants are optimized in the context of the tree in which the building block appears. Building blocks are treated as single units (leaves) during crossover (i.e. are not split).

When searching for a formula $dY/dt = f_i(X_1, \ldots, X_n)$, $Y \in \{X_1, \ldots, X_n\}$, trees in the population are candidate functions for f_i. The fitness of a tree T is calculated as $F(T) = (1 + E(T))^{-1} \times (1 + 0.01 \times \texttt{size\_in\_fit} \times logS(T))^{-1}$, where $E(T) = \sum_{i=1}^{N} (y_i - y_0 - \int_{t_0}^{t_0+ih} T(X_1, \ldots, X_n)dt)^2$, `size_in_fit` (20) is a parameter, and $S(T)$ is the size of the tree T (total number of nodes of T). The integral $\int_{t_0}^{t_0+ih} T(X_1, \ldots, X_n)dt$ is calculated numerically from the measured values of the system variables. The accuracy term dominates the fitness function, but tree size can still have considerable influence in favor of smaller trees.

We use the iterative Levenberg-Marquardt nonlinear optimization method [5] to fit (optimize) the constants in the candidate trees. The total number of iterations used (consumption of computational resources) is fixed and equals `first_iters` $\times$ `pop_size` + `total_iters` ($3 \times 300 + 1000$).

3 Experimental evaluation

Three domains (Population dynamics, Brusselator, Monod) from Džeroski and Todorovski [1] were used. Simulated behaviors were given to GPDD, which was run once for each system variable, producing one differential equation each time. This was repeated 10 times. The parameter settings for GPDD were as described in Section 2. The number of generations was lowered from 30 to 20 and 15 when fast convergence to the correct model was obtained in a preliminary experiment.

Population dynamics. The model of population dynamics consists of two equations: $dN_1/dt = k_1 N_1 - sN_1 N_2$ and $dN_2/dt = sN_1 N_2 - k_2 N_2$.

Ten runs were conducted for both system variables, and the genetic programming algorithm was given 20 generations.

Formulae equivalent to the correct one were found in eight of the ten runs for dN_1/dt. Among these, six are variants of $dN_1/dt = (160 - N_2)(N1/100)$ and have the same fitness. The formulae $dN_1/dt = N_2 N_1(1.6/N_2 - 0.01)$ and $dN_1/dt = (N_1/N_2)(1.6N_2 - 0.01N_2N_2)$ are larger and have smaller fitness. In the remaining two cases, formulae with large error (low fitness) were produced.

In seven of the ten runs for dN_2/dt correct formulae were found, all with the same fitness and of the form $dN_2/dt = 0.01N_2(N_1 - 20)$. In two cases, formulae with large error were produced. The remaining formula has 29 nodes and can be simplified to $dN_2/dt = 0.001 - 0.00005N_1 - 0.2N_2 + 0.01N_1N_2$. It has low error.

Brusselator. The Brusselator is described by the following equations: $dX/dt = A - (B+1)X + X^2Y$ and $dY/dt = BX - X^2Y$.

The ten runs of the genetic programming algorithm were given 15 generations each. The best tree for dX/dt is $dX/dt = X(Y(X-0)-3)+1$, which is equivalent to the first equation in the model. It is worth noting that several formulae have lower error, but much larger size, and consequently lower fitness than the above.

Nine of the ten formulae produced for dY/dt are equivalent to the correct one, three of them being of the form $dY/dt = (2 - YX)X$ and having the highest fitness. For illustration, one of the remaining six formulae has the form $dY/dt = YX(-1 \times X) - (-2 \times X)$, and a fitness of 0.68 (the best formula has fitness 0.72). The tenth formula is both longer and has higher error.

Monod. Equations $dc/dt = -\frac{\mu_{max}}{y}\frac{c}{c+k_s}\,x$ and $dx/dt = (\mu_{max}\frac{c}{c+k_s} - k_d)\,x$ describe the growth of bacteria x given nutrient c.

Ten runs of 30 generations each were conducted. The best formulae for dc/dt and dx/dt are quite complicated and not obviously related to the above equations, although the equation for dx/dt contains the term $\frac{c}{c+100}$ which is the maximum growth rate of the bacteria. They also have relatively high error as compared to the errors of the best equations for previous domains.

The maximum growth rate is a known quantity in ecological modeling and can be used as a background (domain) knowledge in the form of the building block $\frac{c}{c+C}$, where C is a generic constant. Another ten runs of 30 generations each were conducted using the building block $\frac{c}{c+C}$. This time, the two best formulae, $dc/dt = x/(-6/(c/(c+100)))$ and $dc/dt = (x(c/(c+100))) \times (-0.167)$, are equivalent to the correct formula. The three best formulae for dx/dt are also correct. The errors of these formulae are comparable to the ones for the best formulae in the other domains. The building block thus helps significantly towards building more accurate and understandable models.

4 Discussion and further work

As compared to Koza's symbolic regression approach [3], several improvements have been made in our approach. First, parameter identification is carried by the Levenberg-Marquardt method. This gives better parameter values than the entirely evolutionary manipulation of randomly introduced constants done by Koza. Second, our fitness function takes into account the size of the formulae, imposing a bias towards simpler formulae. Finally, our approach allows for the use of background knowledge in the form of building blocks.

In addition, the fitness function takes into account the nature of the problem of discovering dynamics. Namely, instead of numerically introducing derivatives, as done in LAGRANGE [1] (which can produce highly inaccurate results), the fitness function in GPDD integrates the candidate function numerically. Another advantage over the LAGRANGE approach is the more expressive space of models. Furthermore, GPDD avoids the problem of redundancy present in LAGRANGE. Finally, while GPDD can use background knowledge, it is not obvious how this can be done in LAGRANGE.

We applied GPDD to simulated behaviors of several dynamical systems. For the simpler systems, GPDD reconstructed models equivalent to the original even without the use of domain knowledge. For the Monod model, better results were achieved when background knowledge was available. These results illustrate the potential of our approach for discovering dynamics. However, much more experimental evaluation is needed. In particular, a thorough study over a variety of domains, both real and artificial, is needed. In this respect, the sensitivity of GPDD to noisy (erroneous) measurements should be investigated. This is easiest to carry out on artificial data with synthetical noise. An analysis is also needed of the influence of unnecessary (irrelevant) operators and background knowledge (building blocks). Finally, the approach should be applied to real-life domains.

Acknowledgement. This research was supported in part by the Slovenian Ministry of Science and Technology.

References

1. S. Džeroski and L. Todorovski. Discovering dynamics. In *Proc. 10th Int. Conference on Machine Learning*, pp. 97-103. Morgan Kaufmann, San Mateo, CA, 1993.
2. D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, Reading, MA, 1989.
3. J.R. Koza. The genetic programming paradigm: Genetically breeding populations of computer programs to solve problems. In B. Souček, editor, *Dynamic, Genetic, and Chaotic Programming*, pp. 203–321. John Wiley & Sons, 1992.
4. L. Ljung. Modelling of industrial systems. In *Proc. 7th Int. Symposium on Methodologies for Intelligent Systems*, pp. 338–349. Springer, Berlin, 1993.
5. W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes.* Cambridge University Press, Cambridge, MA, 1986.